THE OFFICE OF THE STATE CHIEF INFORMATION OFFICER
ENTERPRISE TECHNOLOGY STRATEGIES

North Carolina Statewide Technical Architecture

Implementation Guidelines:

Application Architecture

STATEWIDE TECHNICAL ARCHITECTURE

Implementation Guidelines: Application Architecture

Initial Release Date:	May 29, 2003	Version:	1.0.0
Revision Approved Date:	Not Applicable		
Date of Last Review:	March 11, 2004	Version:	1.0.1
Date Retired:			
Architecture Interdependencies:			
Reviewer Notes: Reviewed and updated office title and copyright date. Added a hyperlink for the ETS email – March 11, 2004.			

Introduction

The intent of this document is to provide general implementation guidelines within the application technical domain. This will help to ensure that the State of North Carolina adopts uniform and consistent implementations of application solutions across the enterprise.

The key goal of this document is to outline implementation guidelines that, when followed by the solution developers, will lead to a well-designed application solution that has the flexibility to grow with changes in technology and can be maintained in an efficient and effective manner. This is a fundamental principle of the North Carolina Statewide Technical Architecture.

This implementation guild is currently being revised both to better reflect the state of technology and improve upon existing architectural guidelines to better support state agency initiatives.

Implementation Guidelines

Implementation Guidelines for the design and development of applications

Guideline 1: Use a three-tier architecture with access to N-tier shared services.

Rationale

- Large, complex projects that are anticipated to have high usage volumes and/or long life spans will be better served by implementing applications with a three-tier architecture with access to an N-tier service oriented architecture.
- Three-tier client/server applications can be easier to modify to support changes in business rules.
- With three-tier client/server applications, there is less risk in modifying the code which implements any given business rule.
- Three-tier client/server applications can be made to support multiple user interfaces: character, graphical, web browser, telephones, and others.

Implementation Approach for Applications

Avoid New Deployment Migrate from Technology	Current Technology Direction	Emerging Technology
Monolithic applications	Web-enabled front-end. This de-couples the user interface from the other code and allows changing the application architecture without affecting the user. Object-oriented application architecture	
Two-tier applications	Three-tier architecture with access to N-tier shared services. Object-oriented Enterprise Java Bean and servlet application architecture	

Table 1 – Implementation Approach for Applications

Implementation Guidelines for managing applications

Guideline 1: Design Instrumented applications that can report conditions and receive commands.

Rationale

- Manual administration is better than none at all, but adding automatic monitoring to an application is more efficient, and gives the administrator a better opportunity to manage the system effectively.

Guideline 2: Design applications to report resource thresholds so administrators can proactively intervene to prevent applications from failing.

Rationale

- Proactive identification of potential problems causes fewer impacts to the user and is often simpler to resolve appropriately.

Implementation Approach for Designing Manageable Applications

Avoid New Deployment Migrate from Technology	Current Technology Direction	Emerging Technology
Unmanageable applications	Instrumented applications that can report conditions and receive commands.	None at this time
Reporting failures and outages.	Reporting resource thresholds so administrators can proactively intervene to prevent applications from failing.	None at this time

Table 2 - Implementation Approach for Designing Manageable Applications

Implementation Guidelines for Component Reuse

Guideline 1: Copying or linking code into applications should be avoided, in favor of using callable services and object technology.

Rationale

- Copying and linking code into an application are popular but expensive reuse methods because maintenance coding, testing, and debugging must be performed everywhere the code has been used in the application.

Guideline 2: Eliminate code redundancy in applications and between applications by creating and implementing reusable components.

Rationale

- Reusable components that provide common functions can be designed with a common API in order to allow operability with a variety of applications regardless of platforms and operating systems used.
- Legacy applications can be modified to use shared services as they become available.

Implementation Approach for Reusable Components

Avoid New Deployment Migrate from Technology	Technology Direction	Emerging Technology
Copying or linking code into each application	Callable services	Object technology
Redundancy of code in monolithic applications	Reusable components	

Table 3 - Implementation Approach for Reusable Components

Implementation Guidelines for Component Services

Guideline 1: Proprietary security products must be phased out in favor of solutions that use open security protocol standards.

Rationale

- Security solutions usually require integrating several products together rather than implementing a single end-to-end solution. The level of interoperability and flexibility required for achieving adequate security levels can be accomplished only through implementing pieces that support open security protocol standards.

Implementation Approach for Service Components

Avoid New Deployment Migrate from Technology	Technology Direction	Emerging Technology
Proprietary security	Open standards based	Open protocols written for

products	security protocols using SSL, IPSec, S/MIME, GSS-API	Common Data Security Architecture (CDSA)

Table 4 - Implementation Approach for Service Components

Implementation Guidelines for Object-Oriented Components

Guideline 2: Design applications to create new reusable components and to reuse existing reusable services.

Rationale

- Minimize code and data duplication.
- Allows greater adaptability of applications, and lower maintenance costs.
- Sets the stage for using true object-oriented technology.

Implementation Approach for Object-Oriented Componentware

Avoid New Deployment Migrate from Technology	Technology Direction	Emerging Technology
Monolithic design	Service oriented design	Object-oriented design

Table 5 - Implementation Approach for Object-Oriented Componentware

Implementation Guidelines for Software Accessibility

Guideline 1: Provide navigation tools and orientation information in pages to maximize accessibility and usability.

Rationale

Content developers should make content understandable and navigable. This includes not only making the language clear and simple, but also providing understandable mechanisms for navigating within and between pages.

- Not all users can make use of visual clues such as image maps, proportional scroll bars, side-by-side frames, or graphics that guide sighted users of graphical desktop browsers.
- Users also lose contextual information when they can only view a portion of a page, either because they are accessing the page one word at a time (speech synthesis or Braille display), or one section at a time (small display, or a magnified display).
- Without orientation information, users may not be able to understand very large tables, lists, menus, etc.

Implementation Approach for Application Accessibility

Avoid New Deployment Migrate from Technology	Current Technology Direction	Emerging Technology
Inaccessible static pages	Provide navigation tools and orientation information in pages to maximize accessibility and usability.	Accessible Portal-based development

Table 6 - Software Application Accessibility Implementation Strategies